APPLYING SUPERVISED MACHINE LEARNING TECHNIQUES TO MUNICIPAL BOND TRADING

A THESIS
SUBMITTED TO THE FACULTY OF
UNIVERSITY OF MINNESOTA
BY

ROLAND A JACOBUS

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE

FADIL SANTOSA

May 2017

Acknowledgements

I would like to thank my advisor and teacher Fadil Santosa for his patience and encouragement and my wife, Ranee Jacobus, for her unwavering support.

Dedication

This paper is dedicated to my children Megan, Matt, Kendal, and Ryan to remind them that an education is a lifelong pursuit and to the team at ASA LLC who have limitless capacity to take these concepts to higher levels.

Abstract

In this paper we will examine how artificial intelligence or machine learning can be used to make better municipal bond trading decisions. The paper will examine a variety of classification models trained in a supervised environment. The paper will discuss:

1. *How to prepare data for machine learning analysis*
2. *The basic mathematical concepts of each model*
3. *The results of each model and how to interpret them*
4. *How to fine-tune parameters for optimal performance*

Table of Contents

List of Equations

List of Figures

List of Tables

Chapter 1 Municipal Bond Background

The municipal bond market is an over the counter market. Unlike the New York Stock Exchange, bond trades do not occur on a centralized exchange and therefore are not easily observable. There are over 75,000 different municipal bond issuers and over 1,100,000 different bonds in the marketplace. Given the vast array of issuers and bonds, the market structure is not conducive to transparent pricing.

Since bond prices are not visible on a centralized exchange, investors must rely on other means to assess fair purchase and sale prices. Professional investors rely on their sales coverage from financial intermediaries like broker dealers, to keep them abreast of relevant trade levels. Non-professional investors, who lack timely trade information, often make uninformed and off-market investment decisions.

In an effort to give all investors access to the same trade information, the Municipal Securities Rulemaking Board (MSRB) created the Electronic Market Making Access (EMMA) website. The MSRB now requires all financial intermediaries to report their municipal bond trades within ten minutes of trade time. The MSRB aggregates this data and makes it publicly available via the EMMA website.

This paper will use publicly available trade data from EMMA to explore how machine-learning models can help investors make better trading decisions.

Chapter 2 Machine Learning Basics

The term “Machine Learning” describes the process of a computer (machine) learning a mathematical relationship between inputs and outputs. Inputs are the data features like height and weight and outputs are the predictions like male or female. In machine learning terminology, we call this mathematical mapping a model. The goal is for the machine to read a data set and learn a mathematical mapping that is accurate and reliable.

This paper will focus on models that predict a class, not a value. We call these classification models. Predicting the gender from height and weight data is an example of a classification model. In our paper, we will use various details about municipal bond trades to classify them as either winners or losers.

Our dataset includes samples, features, and targets. Each sample is a unique trade, the features include a variety of bond and trade details, and the target is either a winner or loser. Properly training the model requires that we already know if the sample is a winner or loser. We call this learning process, supervised learning. Ideally, if our model can make correct classifications on our training data it will make correct classifications on new data where the classification is not known.

There are many types of classification algorithms. In two-dimensional space, some models find a linear boundary that separates the winners and losers. Some models use statistics/probabilities to determine the most likely classification. Other models find a non-linear boundary that separates the data into their proper classes. Our aim is to find the best one for our particular data set.

Chapter 3 Project Data

As we have suggested above, computers are machines that will read data literally. If for example, we feed the algorithm “five”, it will not infer the number 5, it will read “five” as a five-character text string. If we forget to put the decimal in front of the number 5, it will read the input as 5 and not .5. Our models are only as good as our data and therefore we must pay particular to its accuracy and format.

Gathering Municipal Bond Data

We identified frequently traded bonds and bonds with highly rated credit profiles by searching the Bloomberg municipal bond database. We searched for bonds that met the following criteria:

- Non-callable
- Investment grade
- Deal sizes greater than 25 million
- Tax exempt coupon
- Maturities greater than 10 years

The query resulted in a total of 451 bonds; a small subset of the 1.1 million bonds available in the Bloomberg database.

We then queried the EMMA database for all trades involving these bonds that occurred over a thirty-day period between 1/19/2016 and 2/17/2016. In an effort to identify meaningful trades, we filtered for trades greater than 100,000 dollars. We also filtered all the transactions between dealers as they are often done at off market prices. This query resulted in 615 different trades.

Determining the Target Class

A new bond purchase is done at a price and the price equates to an internal rate of return or yield. At the time of purchase, we compare the purchase yield to a benchmark yield, which represents the yield for all bonds of similar character. The difference between these two yields is called the buy spread. At the end of the thirty day period we evaluate all the bonds in our data set and using a price determined by a third party evaluator, we again calculate the yield and spread to the benchmark. We call this spread the current spread. If the current spread is narrower than the buy spread then the trade is a winner, otherwise a loser.

Data Notation

To describe our machine learning models and learning processes consistently, we will make use of basic linear algebra notation and use matrix and vector notation. The intent of this notation is to simplify our analysis and descriptions.

Our dataset consists of 615 trades and each trade will have 9 features. For those readers familiar with Microsoft Excel, think of the dataset as 615 rows and 9 columns. We will describe the feature data as a 615-by-9 matrix:

Equation 1_Data set notation

$$\mathrm{X} = \begin{bmatrix} \chi_1^{(1)} & \chi_2^{(1)} & \cdots & \chi_9^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ \chi_1^{(615)} & \chi_2^{(615)} & \cdots & \chi_9^{(615)} \end{bmatrix}$$

Each row represents one trade, and each column represents a feature. We will write this as a 9 dimensional row vector:

Equation 2_Trade row notation

$$\boldsymbol{\chi}^{(i)} = \begin{bmatrix} \chi_1^{(i)} & \chi_2^{(i)} & \chi_3^{(i)} & \chi_4^{(i)} & \chi_5^{(i)} & \chi_6^{(i)} & \chi_7^{(i)} & \chi_8^{(i)} & \chi_9^{(i)} \end{bmatrix}$$

Each column represents a different feature. We will write this as a 615 dimensional column vector:

Equation 3_Feature column notation

$$\boldsymbol{\chi}_j = \begin{bmatrix} \chi_j^{(1)} \\ \chi_j^{(2)} \\ \vdots \\ \chi_j^{(615)} \end{bmatrix}$$

The target vector labels each sample as either a winner or loser. We write this as a 615 dimensional column vector:

Equation 4_Target column notation

$$\boldsymbol{Y} = \begin{bmatrix} y_{10}^{(1)} \\ y_{10}^{(2)} \\ \vdots \\ y_{10}^{(615)} \end{bmatrix}$$

Function Notation

As way of further introduction, the machine learning process involves finding boundaries that partition our trades into winners and losers. These boundaries can be linear or nonlinear.

The equation below represents a line in two dimensions. We recall from basic algebra that m describes the slope of the line. A slope that approaches zero describes a horizontal line and a slope that approaches infinity describes a vertical line. The constant b shifts the line vertically relative to the origin.

Equation 5_Function notation for a line in two dimensions

$$y = mx + b$$

Most data sets will consist of many features and therefore the boundary will be a hyper plane. In machine learning terminology, w is used to define the weight of each feature.

Equation 6_Function notation for a hyper plane in multi dimensions

$$y = \ weight_1 \times feature_1 + \ weight_2 \times feature_2 + \ weight_n \times \ feature_n + bias$$

$$y = \ w_1 x_1 + \ w_2 x_2 + \cdots \ w_n \ x_n + bias$$

Chapter 4 Preparing the Data for Evaluation

As we alluded to earlier, if we train the models with bad data types, inconsistent numerical scales, or redundant features, our functions will produce poor results. Preparing the data so that a machine can interpret it is an important and time-consuming task.

Setting Data Types

Our challenge is to convert the various data types (numerical, nominal, date, binary, and string) into a form that our algorithms can interpret.

Numeric Data

Our models are mathematical functions and therefore they handle numeric data well. In Equation 6 we showed that many models rely on the sum product of the feature weights and the feature values. Optimization techniques make small changes to the weights until the prediction error is minimized. If the numeric feature values are of different scale then the largest feature values will tend to have more influence on the output than those with smaller values.

We use two techniques to reduce numerical scale imbalances. For normally distributed data we standardize the data around a zero mean and a unit variance.

Equation 7_Standardization

$$x_{std}^{i} = \frac{x^{i} - \mu_{x}}{\sigma_{x}}$$

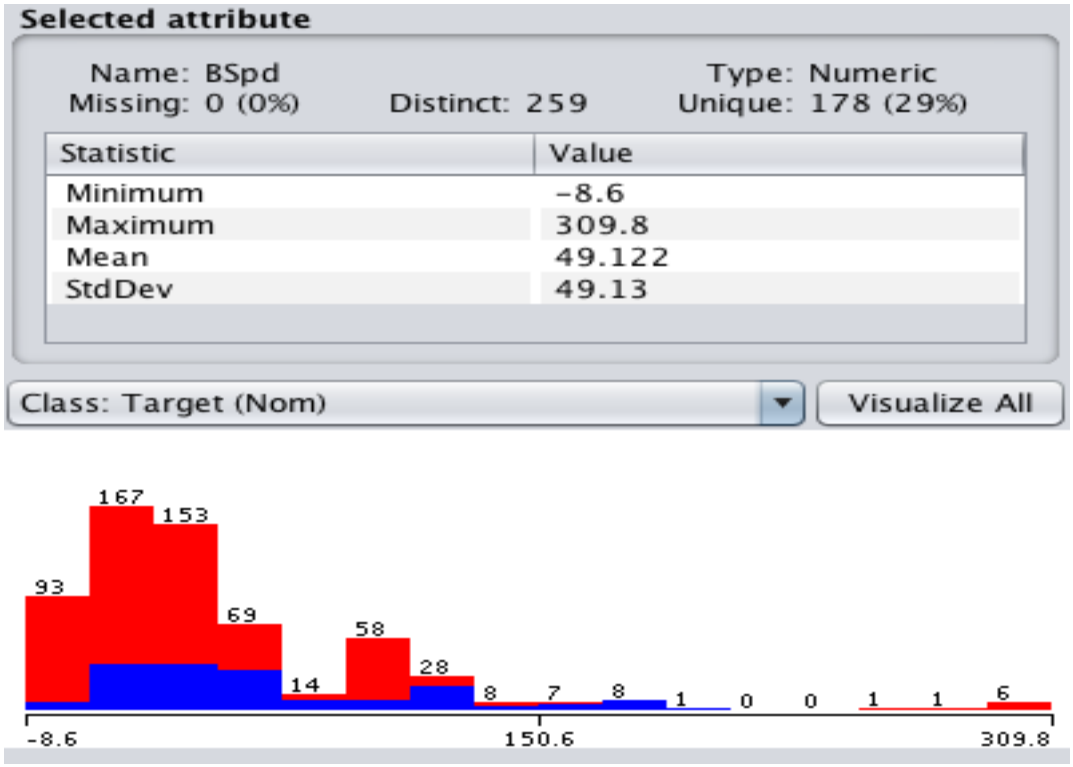
For non-Gaussian data we scale the values between 0 and 1.

Equation 8_ Normalization

$$x_{norm}^{i} = \frac{x^{i} - x_{min}}{x_{max} - x_{min}}$$

The figure below is a screenshot of the bid-spread feature. The distribution is not “bell shaped” or Gaussian and therefore we will normalize this feature to a scale between 0 and 1.

Figure 1_ Non-Gaussian Data



In our project, we will create two data sets, one where the numeric data is normalized and one where the numeric data is standardized. We will examine the model results on both sets of data.

Ordinal

Ordinal features imply order. For example a “large” beverage is usually more expensive than a “small” beverage. In our data set, a triple A rated security is more secure than a single A rated security. We mapped the ratings from different rating agencies to a numeric scale between 0 and 9; a withdrawn rating (wr) to zero and a “none” rating to 1. The BBB rating to 3.5, A rating to 5.5, AA rating to 7.5 and AAA rating to 9.

Moody	Value	SP	Value
WR	0.00	WR	0.00
None	1.00	None	1.00
Baa1	3.00	BBB-	3.00
Baa2	3.50	BBB	3.50
Baa3	4.00	BBB+	4.00
A1	5.00	A-	5.00
A2	5.50	A	5.50
A3	6.00	A+	6.00
Aa1	7.00	AA-	7.00
Aa2	7.50	AA	7.50
Aa3	8.00	AA+	8.00
AAA	9.00	AAA	9.00

Nominal

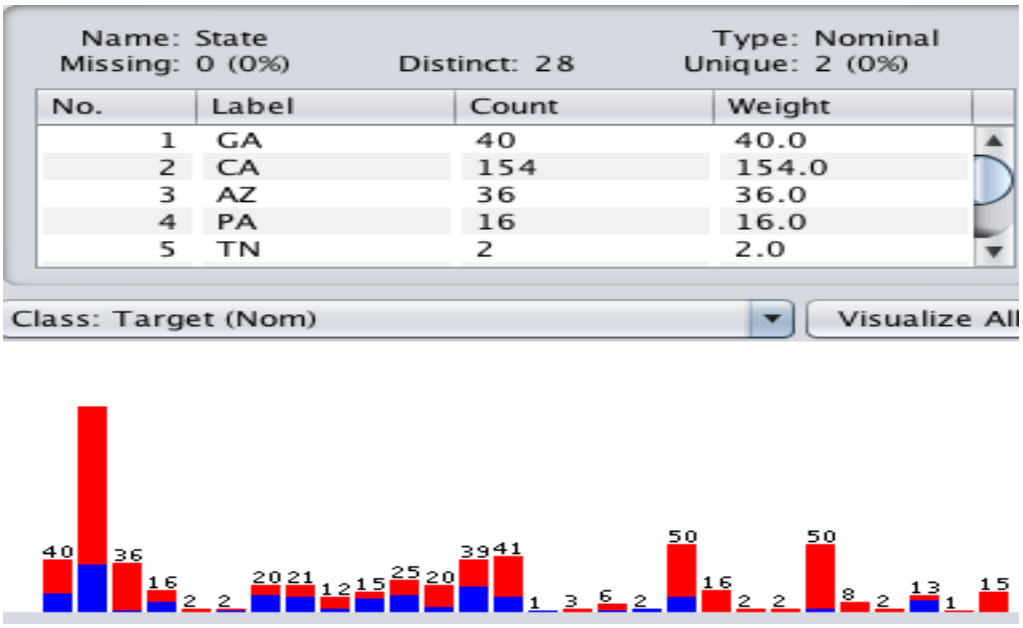
Nominal features do not imply order but do have meaning. For example, the bonds that we are examining are issued from different states. Each state has a different tax rate. All else equal, bonds issued from high tax states are more valuable than bonds issued from low tax states.

For models that utilize a version of Equation 6, the nominal values cannot be used in the sum product calculation. A popular technique is to convert the nominal data to a binary vector. For example, if there are bonds from only two states in our data set (NJ and NY). Then the state field will be a binary vector of length two. Those bonds from NJ will be [1,0] and those bonds from NY will be [0,1].

Statistical models use frequencies and probabilities to make classifications and technically do not need to convert the nominal features to binary vectors. However, most statistical machine learning software packages convert the binary vectors to frequencies so we will convert all nominal data.

As an example, the State field includes 28 different States. We will convert the state field to binary vectors of length 28.

Figure 2_ State Distribution



Removing Outliers

Outliers will skew most models, particularly statistical models. In many cases, especially if parsing is involved, raw data will include categorical data that is misspelled or numeric data that is missing a decimal place.

Example: A trade registered a zero buy yield, which we removed.

Replacing Missing Data

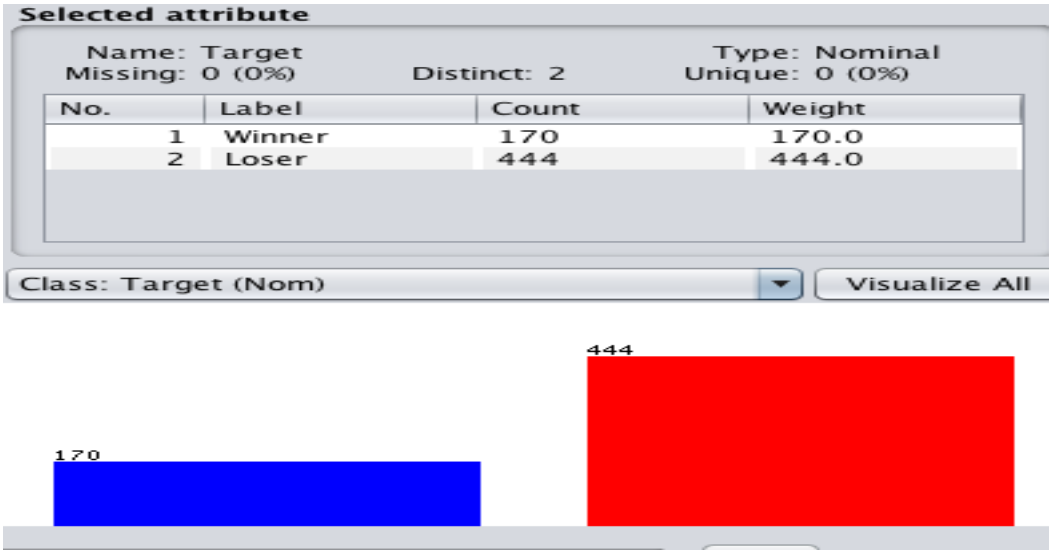
In some cases, a feature may be missing from a particular sample. Since all sample data is valuable it is often better to replace the missing feature than eliminating the entire sample. Replacing the missing data with the mean or median value or in the case of categorical fields, the most frequent is a popular technique.

Example: In our data, we eliminated all incomplete trades.

Creating Balance

Models trained on a data set that only includes winners will not know how to classify losers. In an ideal world our data should be balanced and have an equal number of winners and losers. Our data includes 170 winners (28%) and 444 losers (72%).[1]

Figure 3_Data Balance



Unbalanced data is the norm and a common approach is to acknowledge the imbalance but leave the imbalance in the training data.

Other techniques require some data manipulation:

Under Sampling Techniques

Techniques used to reduce the majority class samples. These techniques are best used for large data sets. Since our data set is small this is not appropriate.

Over Sampling Techniques

Techniques used to increase the minority class. We will use random resampling with replacement to increase the minority class and balance the data set.

We will test our models on both the imbalanced data set and a balanced set using the oversampling technique. When viewed in the results sections, B will designate a balanced data set and IB will designate an imbalanced data set.

Feature Selection

In machine learning speak there are two types of model error, bias and variance.

Let’s assume our dataset includes all collegiate basketball players and we want to train a classification model to identify those players most likely to play in the NBA.

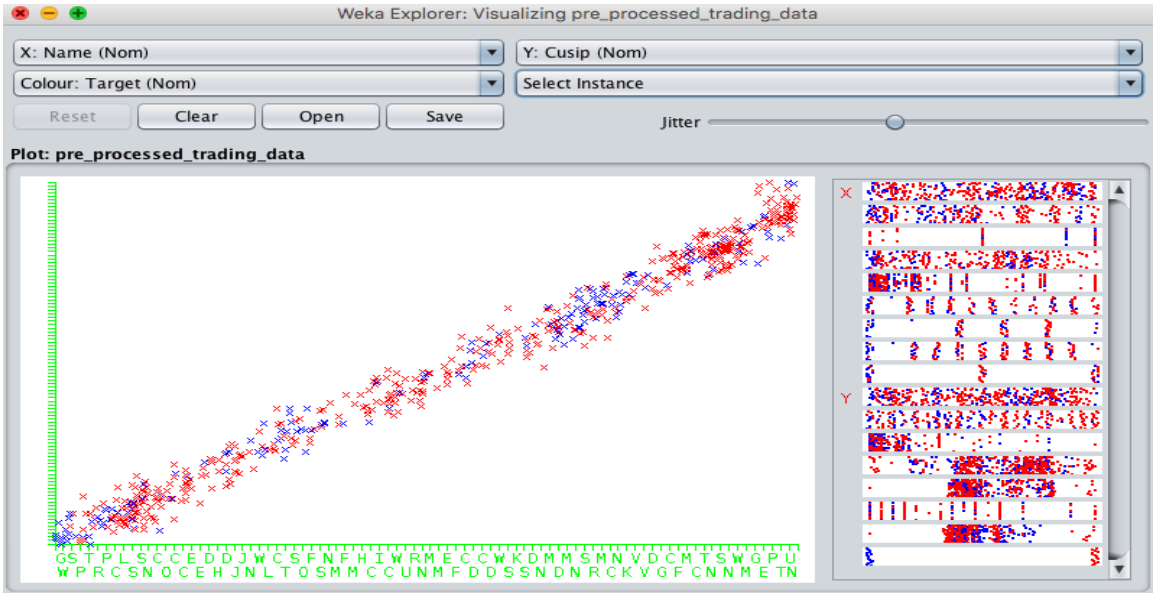
If we train our model only using the height feature it will likely classify the tallest players as NBA players. Our model bias favors taller players, which may not always be the case. On the other hand, if we include a variety of features including their University, then the model will likely favor those players from the most recent basketball powerhouses. Since these powerhouses change every few years, our model results will vary depending on the year we collect the data. We call this variance.

Our goal is to find a balance between bias and variance. We can do this by eliminating redundant data and training on features that are correlated to our target class.

Redundancy

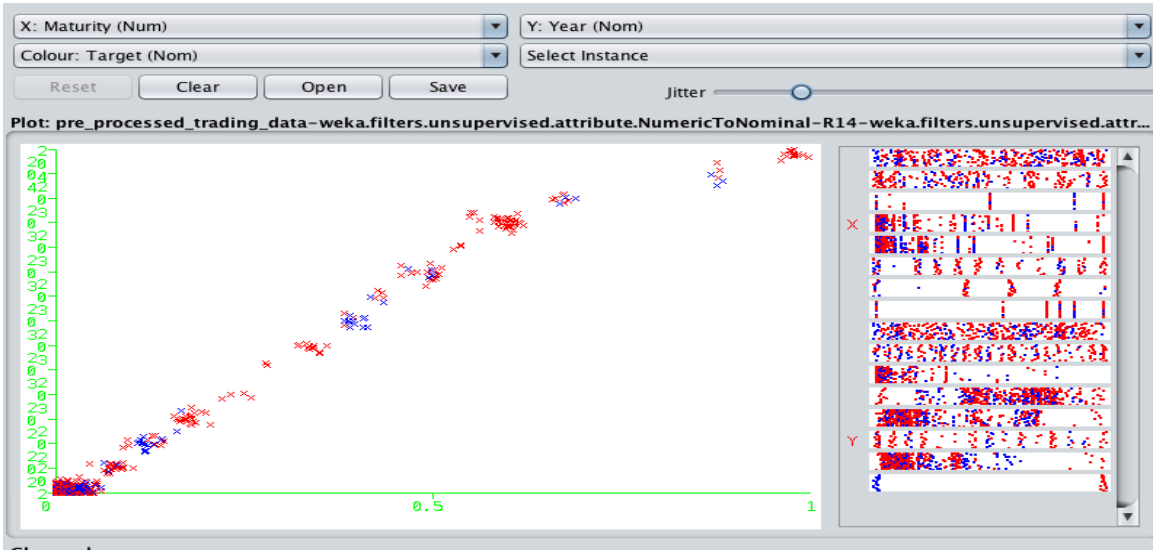
Redundant data features tell our model the same thing. These features have a high degree of correlation to one another. As you can see below the fields “Name” and “Cusip” are highly correlated so we do not need both of them.

Figure 4_Name versus Cusip



The fields “Maturity” and “Year” are also strongly correlated and one can be removed from the sample set.

Figure 5_Maturity versus Year



We found the following fields to be redundant and removed them: Name, Maturity, Cusip, Price, and Bid Yield.

Relevance

We use three different techniques to determine the features most relevant to classifying winners and losers.

Correlation

We rank the Pearson's correlation coefficient between each feature and the target class and disregard those features that show little correlation.

Information Gain

Information gain measures each feature's ability to split the data into "pure" subsets. For example, a feature that splits the data into 10 winners and 0 losers is ranked higher than a feature that splits the data into 5 winners and 5 losers. This technique ranks the features by the information gain and we disregard the lowest ranked features.

Learner Techniques

We pick a generic classification model (a decision tree model) and run the model with different subsets of features. The subsets that produce the best performance are the features selected.

Table 1_Attribution Selection Techniques

Data	Correlation	Info Gain	Learner
Imbalanced	Bspd, Ind, Typ,Yr, St	Iss, Bspd, Yr,Ind, St	St, Cpn, Iss,Ind, Typ
Balanced	Bspd, St, Ind,Yr, Typ	Bspd, Iss,St,Ind, Yr	St, Cpn, Iss,Yr, Bspd

Based on the analysis above, the fields BSpd, State, Industry, Type, Year, Issue Size, and Coupon appear to be most relevant; Rating, Amount, and Trade Date are not relevant.

Our original dataset included 15 features. We removed Name, Maturity, Cusip, Price, Bid Yield , and Trade Date, leaving us with 9 features.

Our final feature set includes:

1. ***State*** - Municipal bonds pay a coupon that is exempt from federal taxes and in most instances; the coupon is exempt from state taxes. All else equal, bonds from high tax states are more valuable than bonds from low tax states.
2. ***Coupon***- The coupon is the tax-exempt rate issuers pay to bond holders.
3. ***Year***- The year that the investor receives his principal back.
4. ***Type*** – The general type of legal structure that supports the revenue stream.
5. ***Industry*** - The specific revenue source that the issuer uses to pay the debt service.
6. ***Issue Size*** - The total amount of bonds in the loan.
7. ***Amount*** – The amount is the trade size (in dollars) of each trade
8. ***Rating*** – The rating represents the likelihood that the issuer will pay debt service on a timely basis
9. ***Buy Spread*** - The difference between the purchase yield to maturity and the risk free tax-exempt rate to maturity.

Chapter 5 Tools for Training Models

Training Techniques

Generally speaking a data set is split into a training set and a test set. The training set is used to train the model and the test set is used to test it. Randomly partitioning 80% of the samples to the training set and 20% to the test set is typical.

For smaller data sets like ours, cross validation techniques are more common. This technique splits the data set into folds (or equal subsets) with one fold designated as the test set and the remaining folds as training sets. The idea is to change the test set after each iteration and to average the results to get a more balanced perspective.

Optimization Techniques

Intuitively, each model that we train will not perfectly match inputs to outputs. The difference between the predicted output and the actual output is called the error or cost. Optimization techniques tweak the model until the cost is minimized.

Gradient descent is a popular and powerful technique we will use to find the feature weights that minimize this cost. The gradient is a vector of partial derivatives and represents the change in the error given a change in the feature weights.

Recall that as the derivative or gradient approaches zero the function approaches a minimum or maximum. The gradient descent technique uses the derivative to determine the direction the weights need to be adjusted to reach this minimum.

Mathematical Description

Stochastic gradient descent updates the function coefficients after every sample in a direction that reduces the error on the next prediction for that sample. The coefficients are adjusted by the derivative of the cost function times a learning rate:

Equation 9 _Gradient Descent Weight Adjustments

$$w_{t+1}^{i} = w_{t}^{i} - \alpha\delta$$

$$\alpha\ (alpha) = learning\ rate\ or\ step\ size$$

$$\delta\ (delta) = derivative\ or\ gradient\ of\ cost\ function$$

Computation Process[2]

1) *Initiate values of the function weights at 0.*
2) *Calculate the model prediction for sample* (x^{i})
3) *Calculate the new derivative of the cost function* δ
4) *Update the weights:* $w_{t+1}^{i} = w_{t}^{i} - \alpha\delta$.
5) *Repeat this process for all samples. One time through all samples is called an epoch.*
6) *Repeat this process until the error is less than a threshold*

Chapter 6 Measuring Model Performance

Our data consists of 72% losing trades (losers) and 28% winning trades (winners). As discussed earlier, a model that simply predicts the dominant class (losers) will be 72% accurate. If we evaluate the model strictly by the accuracy rate (72%) then the model appears to be a good one. However, in this example, the model does not classify any winners correctly and if it cannot help our clients find winning trades then it is not a good one. The accuracy rate is one way to measure model performance, but given the example above, we need better way.

We will use a confusion matrix to evaluate our models. Some say the confusion matrix is "confusing" but it's actually quite simple. The rows represent the actual classifications of the trades and the columns represent the predicted classifications. If our model classifies with 100% accuracy it will look like this:

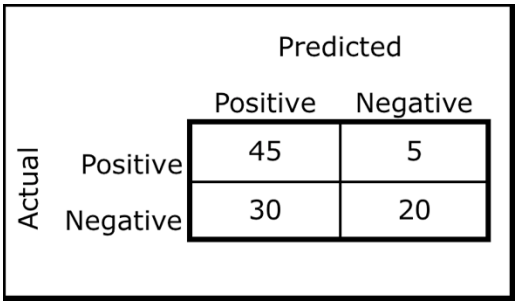
Figure 6_Confusion Matrix

	Predicted: Winner	Predicted: Loser	Total
Actual: Winner	True Winner 170	False Winner 0	170
Actual: Loser	False Winner 0	True Loser 444	444
Total	170	444	614

A true classification means the model classified correctly and a false classification means the model classified incorrectly.

We will use the hypothetical confusion matrix below to become familiar with the terminology and various accuracy measurements. The labels Positive and Negative are used to describe classes, in our case, "Winners" and "Losers". The hypothetical matrix consists of 50 samples in the positive class and 50 in the negative class. Our model predicted 75 to be in the positive class and 25 in the negative class.

Figure 7_Confusion Matrix Calculation Example



Accuracy Rate

Accuracy – measures the model's correct classifications relative to total classifications.

$$\frac{45 + 20}{100} = 65\%$$

Error – 1 minus the Accuracy Rate

Class Accuracy

Class accuracy measures how accurately the model classifies each class. It is important to know if the model is better at classifying winners or losers.

True Winner Rate (Sensitivity) – measures the model's ability to classify the Winners; predicted Winners relative to the total number of Winners.

	Predicted Win	Predicted Loss		Win Accuracy
Actual Win	**45**	5	50	$\frac{45}{45 + 5} = 90\%$
Actual Loss	30	20	50	
	75	25		

True Loser Rate (Specificity) – measures the model's ability to classify the Losers; predicted Losers relative to the total number of Losers.

	Predicted Win	Predicted Loss		Loss Accuracy
Actual Win	45	5	50	$\frac{20}{30 + 20} = 40\%$
Actual Loss	30	**20**	50	
	75	25		

This hypothetical model predicts winners more accurately than losers.

Class Reliability

Class reliability measures the reliability of the classifications. If our model predicts a winner and it's a loser then our customers lose money.

Winner Reliability (Precision) – measures the correct Winner classifications relative to all of the Winner classifications.

	Predicted Win	Predicted Loss		Win Reliability
Actual Win	**45**	5	50	$\frac{45}{45+30} = 60\%$
Actual Loss	30	20	50	
	75	25		

Loser Reliability (Recall) – measures the correct Loser classifications relative to all of the Loser classifications.

	Predicted Win	Predicted Loss		Loss Reliability
Actual Win	45	5	50	$\frac{20}{5+20} = 80\%$
Actual Loss	30	**20**	50	
	75	25		

This hypothetical model predicts losers more reliably than winners.

F1 (Wikepedia) Measure

The F measure is a weighted average of the accuracy and reliability measures.

$$F1\ (Win) = \frac{2 \times 45}{2 \times 45 + 5 + 30} \quad \frac{90}{125} = 72\%$$

$$F1\ (Loss) = \frac{2 \times 20}{2 \times 20 + 5 + 30} \quad \frac{40}{75} = 53\%$$

The measurements above suggest that although the model classified 90% of the "winners" correctly, it did not classify them very reliably (only 60% of the winning predictions were correct).

For each model we will present a table with the following measurements:

Table 2_Sample Result Template

Class	Accuracy	Reliability	F1 Stat
Winners	90%	60%	72%
Losers	40%	80%	53%

Chapter 7 Algorithms

There are two types of models that we will evaluate; linear models find linear boundaries that separate trades into winners and losers and non-linear models find non-linear boundaries that separate the trades We will use the WEKA GUI to test our models and in some cases Microsoft Excel to demonstrate how the model's work.

Linear Models

Perceptron Model

In 1943, Warren McCulloch and Walter Pitts published a paper called "A Logical Calculus of the Ideas Immanent in Nervous Activity". They described a neuron as a threshold gate. This neuron (gate) fired an output only after it accumulated signals (from the nervous system) that surpassed a certain threshold. Soon after, Frank Rosenblatt (1957) published the perceptron learning rule based on McCulloch-Pitts neuron model. Rosenblatt intended to use the model for image recognition but the media soon suggested that this new artificial intelligence would soon walk, talk, write and reproduce itself. Needless to say, the model fell short of the hype but we will use this simple model to introduce many of the basic machine learning concepts.[3]

Perceptron Mathematical Description

As a neuron reads signals from the nervous system, a perceptron reads an input vector. The vector is composed of relevant features. In our case, those features include the bid spread, industry, type, state etc. Each of these features is given a weight and the sum-product of these inputs is called the output. If the output is greater than a threshold, the model makes one classification and if it is less than the threshold it makes another classification.

The perceptron reads the inputs. Let x be the input vector of features and w a weight vector, both of length d.

Equation 10_Perceptron Vectors

$$x^i = \begin{bmatrix} x_0^i & x_1^i & \dots x_d^i \end{bmatrix}$$

$$w^i = \begin{bmatrix} w_0^i & w_1^i & \dots w_d^i \end{bmatrix}$$

The perceptron calculates the inner product of these two inputs. The inner product s is the product of the two vectors.

Equation 11_Perceptron Output

$$s = w_0x_0 + w_1x_1 + w_2x_2 + w_dw_d = w^Tx$$

$$bias = w_0 \; so \; we \; will \; set \; x_0 = 1$$

The perceptron's threshold gate $g(s)$ is a binary activation function. If we set our threshold at zero then the perceptron will classify a Winner if $s \geq 0$ and a Loser if $s < 0$.

Equation 12_Perceptron Threshold Gate

$$g(s) = \begin{cases} +1, & s \geq 0 \\ -1, & s < 0 \end{cases}$$

Perceptron Learning Process

The training data consist of n vectors with i features x_i^n, each with a label t_i equal to 1 or -1. In our case, there are 614 vectors each with 9 features, and winners are mapped to 1 and losers to -1.

The goal is to determine the weight vector that results in the fewest classification errors. We will iterate thru each trade, adjusting the weights, and keeping a running total of the squared error. We repeat this process for a set number of epochs or until the error falls below a set threshold.

Equation 13_Perceptron Error

$$\sum_{i=1}^{n} (t_i - g(s))^2$$

We can minimize this function using an iterative process:

1. *Initialize the weight vector and the bias with zero values*
2. *Loop thru all trades (epoch) five times.*
3. *For each epoch, loop thru each trade.*
    a. *Calculate s*
        i. $s = w_0^i x_0^i + w_1^i x_1^i + \cdots w_d^n x_d^n + bias$
    b. *Calculate g(s)*
        i. $g(s) = \begin{cases} +1, & s \geq 0 \\ -1, & s < 0 \end{cases}$
    c. *Calculate the error and keep a running total of the squared errors for each epoch. If the sum of the squared errors for an epoch is zero then stop.*
    d. *Adjust the weights*
        i. $w_{t+1} = w_t + \alpha\left(t^i - g\left(s^i\right)\right)x_t$

The parameter α is the learning rate and is set by the user. This parameter governs the step size that the model uses to adjust the weights. The bias, w_0, is calculated by setting $x_0\ always\ equal\ to\ 1$.

A small learning rate will move the weights ever so slightly and will increase the convergence time while a large learning rate may result in the model jumping over the optimal weight size.

To demonstrate, let's assume a feature $x = 1$ and a learning rate $\alpha = .5$. If our model predicts negative 1 instead of the target, positive 1, then intuitively the weights need to increase so that the output s is greater than zero and the model predicts 1 at the next iteration.

As you can see below, the model works as we expect and adjusts the weight higher.

Equation 14_Perceptron Weight Adjustment Example

$$\Delta w_j = \alpha(t^i - g(s^i))x_j.$$

$$\Delta w = .5(1 - -1)1 = .5\ (2) = 1$$

Perceptron Calculation

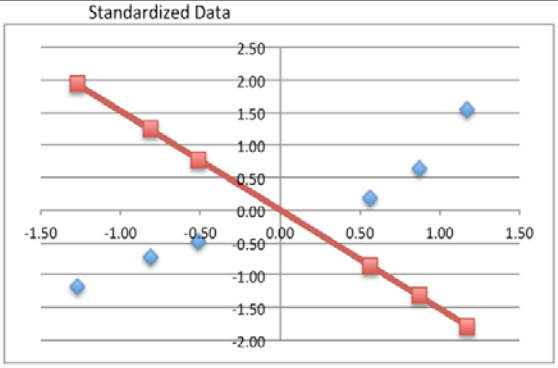
The hypothetical example below demonstrates how the weights and boundary is calculated. After one epoch the function learned optimal weights such that the activation function classified samples with 100 percent accuracy.

The red line visually demonstrates the linear boundary; signals above this line are classified to the negative class and below are classified to the positive class.

Table 3_Perceptron Excel Calculations

X_1	X_2	Class
-0.82	-0.72	1
-1.28	-1.17	1
-0.51	-0.49	1
0.87	0.64	-1
0.56	0.19	-1
1.17	1.55	-1

Standardized Data



Threshold Line

X_1	X_2
-0.82	1.24
-1.28	1.93
-0.51	0.77
0.87	-1.32
0.56	-0.85
1.17	-1.78

	X_1	X_2	W_0	W_1	W_2	s	g(s)	Y	Error	α
1	-0.82	-0.72	0.000	0.000	0.000	0.000	-1	1	2	0.01
	-1.28	-1.17	0.020	-0.016	-0.014	0.058	1	1	0	0.01
	-0.51	-0.49	0.020	-0.016	-0.014	0.035	1	1	0	0.01
	0.87	0.64	0.020	-0.016	-0.014	-0.003	-1	-1	0	0.01
	0.56	0.19	0.020	-0.016	-0.014	0.008	1	-1	-2	0.01
	1.17	1.55	0.000	-0.028	-0.018	-0.061	-1	-1	0	0.01
2	-0.82	-0.72	0.000	-0.028	-0.018	0.036	1	1	0	0.01
	-1.28	-1.17	0.000	-0.028	-0.018	0.056	1	1	0	0.01
	-0.51	-0.49	0.000	-0.028	-0.018	0.023	1	1	0	0.01
	0.87	0.64	0.000	-0.028	-0.018	-0.036	-1	-1	0	0.01
	0.56	0.19	0.000	-0.028	-0.018	-0.019	-1	-1	0	0.01
	1.17	1.55	0.000	-0.028	-0.018	-0.061	-1	-1	0	0.01

Perceptron Results

Table 4_Perceptron Results Summary Table (Balanced Set)

Class	Accuracy	Reliability	F1 Stat
Winners	78.5%	79.8%	79.1%
Losers	80.1%	78.8%	79.5%

Figure 8_WEKA Perceptron Result Output

Balanced Sample Set

```
Correctly Classified Instances         487               79.316  %
Incorrectly Classified Instances       127               20.684  %
Kappa statistic                          0.5863
Mean absolute error                      0.2143
Root mean squared error                  0.4168
Relative absolute error                 42.8552 %
Root relative squared error             83.3564 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.785    0.199    0.798      0.785    0.791      0.586    0.837     0.806     Winner
                 0.801    0.215    0.788      0.801    0.795      0.586    0.837     0.793     Loser
Weighted Avg.    0.793    0.207    0.793      0.793    0.793      0.586    0.837     0.799

=== Confusion Matrix ===

   a   b   <-- classified as
 241  66 |   a = Winner
  61 246 |   b = Loser
```

Imbalanced Sample Set

```
Correctly Classified Instances         513               83.5505 %
Incorrectly Classified Instances       101               16.4495 %
Kappa statistic                          0.5509
Mean absolute error                      0.1706
Root mean squared error                  0.3869
Relative absolute error                 42.5665 %
Root relative squared error             86.4732 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.565    0.061    0.780      0.565    0.655      0.563    0.776     0.705     Winner
                 0.939    0.435    0.849      0.939    0.892      0.563    0.776     0.846     Loser
Weighted Avg.    0.836    0.332    0.830      0.836    0.826      0.563    0.776     0.807

=== Confusion Matrix ===

   a   b   <-- classified as
  96  74 |   a = Winner
  27 417 |   b = Loser
```

Logistic Regression

Alphonse Quetelet and his student, Francois Verhulst, introduced the logistic equation in the 19th century to describe human population growth. In 1920, Raymond Pearl and Lowel Reed, both from Johns Hopkins University, used the function to estimate population growth for all living populations. Joseph Berkson introduced the logistic function to statistics in the 1950s. The function maps any real valued number to a value between 0 and 1. This value represents the probability of the default class. (Cramer, 2003)[4]

Logistic Regression Mathematical Description

Just like the perceptron model, we let x be a feature vector and w a weight vector, both of length d, and s their sum product.

Equation 15_Logistic Regression Mathematical Description

$$x^i = \begin{bmatrix} x_1^i & x_2^i & \dots x_d^i \end{bmatrix}$$

$$w^i = \begin{bmatrix} w_1^i & w_2^i & \dots w_d^i \end{bmatrix}$$

$$s = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_d w_d = w^T x$$

$$by\ setting\ \ x_0 = 1, w_0\ becomes\ our\ bias$$

The logit function maps output s to a probability p; the probability that the sample is in the default class.

Equation 16_Logistic Regression Logit Function

$$p = \frac{1}{1 + e^{-s}}$$

The probability p is passed to a binary function, which predicts the default class 1 if the probability is greater than or equal to .5.

Equation 17_Logistic Regression Binary Function

$$g(p) = \begin{cases} 1, & s \geq .5 \\ 0, & s < .5 \end{cases}$$

Logistic Regression Learning Algorithm

Just like the Perceptron Model, the training data consist of n vectors with i features x_i^n, each with a label t_i equal to 1 or 0 (not 1 or -1).

The goal of the training is to determine the weight vector $[w_0, w_1 \cdots w_d]$ that results in the fewest classification errors.

For each sample, we find the error by finding the difference between the actual classification and the predicted classification. The sum of the errors across all samples is called the cost. Our goal is to minimize the cost function (see Equation 12).

We can minimize this function using an iterative process:[5]

1. *Initialize the weight vector and the bias with zero values*
2. *Loop thru all trades (epoch) five times.*
3. *For each epoch, loop thru each trade.*
    a. *Calculate s*
        i. $s = w_0^i + w_1^i x_1^i + w_2^i x_2^i + \cdots w_d^n x_d^n$
    b. *Calculate the logit*
        i. $p = \frac{1}{1+e^{-s}}$
    c. *Calculate g(s)*
        i. $g(s) = \begin{cases} 1, & p \geq .5 \\ 0, & p < .5 \end{cases}$
    d. *Calculate the error and keep a running total of the squared errors for each epoch. If the sum of the squared errors for an epoch is zero then stop.*
    e. *Adjust the weights*
        i. $\Delta w_t = \alpha(t-p)(p)(1-p)x_t$
        ii. $w_{t+1} = w_t + \Delta w_t$

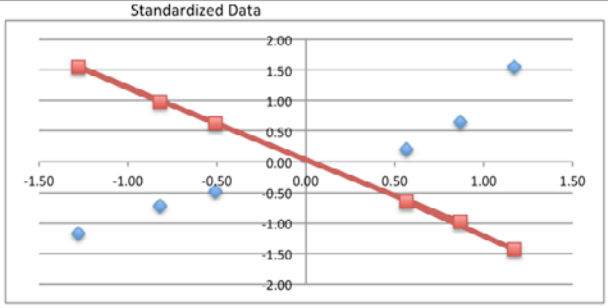
The parameter α is the learning rate and is set by the user. This parameter governs the step size that the model uses to adjust the weights. A small learning rate will move the weights ever so slightly and will increase the convergence time while a large learning rate may result in the model jumping over the optimal weight size.

Logistic Regression Calculation

After one epoch the function learned optimal weights such that the activation function classified samples with 100 percent accuracy.

Table 5_Logistic Regression Excel Calculations

X_1	X_2	Class
-0.82	-0.72	0
-1.28	-1.17	0
-0.51	-0.49	0
0.87	0.64	1
0.56	0.19	1
1.17	1.55	1
68.33	151.67	mean
6.53	44.01	std



Threshold Line

X_1	X_2
-0.82	0.98
-1.28	1.55
-0.51	0.63
0.87	-0.99
0.56	-0.64
1.17	-1.44

	X_1	X_2	W_0	W_1	W_2	s	logit	g(s)	Y	Error	α
1	-0.82	-0.72	0.000	0.000	0.000	0.000	0.500	1	0	-1	0.5
	-1.28	-1.17	-0.063	0.051	0.045	-0.180	0.455	0	0	0	0.5
	-0.51	-0.49	-0.119	0.123	0.111	-0.236	0.441	0	0	0	0.5
	0.87	0.64	-0.173	0.151	0.138	0.046	0.512	1	1	0	0.5
	0.56	0.19	-0.112	0.204	0.177	0.036	0.509	1	1	0	0.5
	1.17	1.55	-0.051	0.238	0.189	0.522	0.628	1	1	0	0.5
2	-0.82	-0.72	-0.007	0.289	0.256	-0.428	0.395	0	0	0	0.5
	-1.28	-1.17	-0.055	0.328	0.290	-0.814	0.307	0	0	0	0.5
	-0.51	-0.49	-0.087	0.369	0.329	-0.438	0.392	0	0	0	0.5
	0.87	0.64	-0.134	0.393	0.352	0.434	0.607	1	1	0	0.5
	0.56	0.19	-0.087	0.434	0.382	0.229	0.557	1	1	0	0.5
	1.17	1.55	-0.032	0.465	0.392	1.122	0.754	1	1	0	0.5

Logistic Regression Results

Table 6_Logistic Regression Results (Balanced Set)

Class	Accuracy	Reliability	F1 Stat
Winners	74.9%	76.9%	75.9%
Losers	77.5%	75.6%	76.5%

Table 7_Weka Logistic Regression Output

Balanced Sample Set

```
Correctly Classified Instances         468               76.2215 %
Incorrectly Classified Instances       146               23.7785 %
Kappa statistic                          0.5244
Mean absolute error                      0.2778
Root mean squared error                  0.3944
Relative absolute error                 55.5549 %
Root relative squared error             78.8828 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.749    0.225    0.769      0.749    0.759      0.525    0.854     0.810     Winner
                 0.775    0.251    0.756      0.775    0.765      0.525    0.854     0.863     Loser
Weighted Avg.    0.762    0.238    0.762      0.762    0.762      0.525    0.854     0.837

=== Confusion Matrix ===

   a   b   <-- classified as
 230  77 |   a = Winner
  69 238 |   b = Loser
```

Imbalanced Sample Set

```
Correctly Classified Instances         517               84.202  %
Incorrectly Classified Instances        97               15.798  %
Kappa statistic                          0.5851
Mean absolute error                      0.216
Root mean squared error                  0.3436
Relative absolute error                 53.8925 %
Root relative squared error             76.7792 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.635    0.079    0.755      0.635    0.690      0.589    0.867     0.745     Winner
                 0.921    0.365    0.868      0.921    0.894      0.589    0.867     0.926     Loser
Weighted Avg.    0.842    0.286    0.837      0.842    0.838      0.589    0.867     0.876

=== Confusion Matrix ===

   a   b   <-- classified as
 108  62 |   a = Winner
  35 409 |   b = Loser
```

Linear Model Conclusion

Using the balanced sample set, the Logistic Regression model classified with 76.2% accuracy versus 79.6% for the Perceptron model. The Perceptron model classified the winners more reliably as well 79.8% versus 76.9%.

Using the F1 stat as our criterion, the Perceptron model is a superior model for our data.

Table 8_F1 Stat Comparison for Linear Models

Class	Perceptron	Logistic
Winners	79.1%	75.9%
Losers	79.5%	76.9%

Non-Linear Models

Non-linear data cannot be separated with a straight line. Most data falls into this category. In this section we will evaluate models that can classify non-linear data.

Naïve Bayes Model

The Naïve Bayes model is a binary and multiclass classification model that uses class probabilities and conditional probabilities to classify target data. At the core of the model is Bayes's Theorem named after Thomas Bayes.

The Reverend Thomas Bayes introduced Bayes's theory in the 1700s. Bayes's believed that an event's future probability is based on prior knowledge. Laplace formulized the theory in the 1800's. In words, the probability of event A given event B is equal to the probability of event B given event A times the probability of event A, all divided by the probability of event B. For over a hundred years, most mathematicians trivialized the theorem's validity because it relied on prior knowledge; in most cases a "hunch" and not "fact". [6]

As further background, let's imagine a family gathering during the Christmas holiday. Mom is in the kitchen and there is warm aroma of chocolate chip cookies waffling in the air. Most would say with 100% certainty that Mom is baking fresh chocolate chip cookies. However, what if Aunt Sue, who worked at Mrs Fields cookie store, was visiting with Mom in the kitchen. Are we still 100% certain the aroma is coming from Mom's fresh baked cookies or is their a chance Aunt Sue brought a fresh baked dozen from Mrs Fields?

The probability that Mom baked cookies is based on prior probabilities; how often does Mom bake cookies during the holidays P (event A), how often do we smell the aroma of fresh cookies during the holidays P (event B), and how often do we smell the aroma when Mom is baking cookies P (B|A). Since we know there are times when Aunt Sue brings fresh baked cookies tand we know that Aunt Sue is visiting in the kitchen then the probability that Mom is baking the cookies is less than 100%.

Not until the recent computer revolution in the 1980s has the theorem been embraced by modern day scientists and statisticians. Most commonly used to discern email spam, the theorem is now widely used to drive driverless cars, predict election outcomes, and help diagnose diseases.

Mathematical Description

Bayes theorem gives us a framework to calculate the probability of a class given prior knowledge about the features in our dataset and observed data. We call this posterior probability.

As an example, let's assume that our data consists of 20 bond trades. Ten of the bond trades are California bonds and all of them are winners. The other ten are New York bonds and three are winners and seven are losers. If the bond is a winner, what is the probability it is a New York bond?

Equation 18_Bayes Theorem

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

The event A is Class = winner the event B is State = New York. The probability of a winner is 13/20 or 85%, the probability a bond is from New York is 10/20 or 50%, and the probability of a bond from New York being a winner is 3/10 or 30%. Plugging these inputs into Bayes' formula:

Equation 19_Bayes Theorem Example

$$P(A|B) = \frac{30\% \times 85\%}{50\%} = 51\%$$

Formally,

$P(A|B)$ - Posterior Probability ➔ Probability of class given a feature.

$P(B|A)$ - Conditional Probability ➔ Probability of feature given class.

$P(A)$ - Prior Probability ➔ Probability of class.

$P(B)$ - Prior Probability ➔ Probability of feature.

The denominator is used to normalize the probability and will be the same across all data points and therefore we can drop it from our calculations.

Equation 20_Bayes' Theorem Applied to Classification

$$p(class|feature) = p(feature|class) \times p(class)$$

If there is more than one feature then we assume that they are independent and the posterior probability is calculated as follows[7]:

Equation 21 _Bayes's Theorem with Multiple Classes

$$p(class|feature) = \left[\prod_{1}^{i} p(feature_i \mid class)\right] \times p(class)$$

Our model will use the training data to determine the posterior probabilities for all features. The model will classify new data to the class with the maximum posterior probability or maximum a posteriori (MAP).

Algorithm and Learning Process

1. *Calculate the class probabilities for each class*
   a. $class\ probability \rightarrow p_k = \frac{count\ (t_k)}{\sum_1^n count(t_n)}$
2. *Calculate the conditional probabilities for each feature.*
   a. $conditional\ probability\ \rightarrow p(feature_i|class_k) \rightarrow\ \frac{count\ of\ (feature_i\ \wedge class_k)}{count\ of class_k}$
3. *Calculate the posterior probabilities for each feature.*
   a. $\left[\prod_1^i p(feature_i|class_k)\right]\ \times\ p_k$
4. *Classify each sample to the maximum posterior probability.*

Data Preparation

The Naïve Bayes model was originally intended for categorical data where we use simple counts and frequencies to calculate the probabilities. This model can be extended to numerical values by using the Gaussian Probability Density Function (PDF) to calculate the probabilities.

Equation 22_Probability Density Function

$$pdf\left(x_i, mean(x), std(x)\right) = \frac{1}{\sqrt{2\pi} \times std} \times e^{\frac{-(x-mean^2)}{2 \times std^2}}$$

Results

Table 9_Naïve Bayes's Model Results

Class	Accuracy	Reliability	F1 Stat
Winners	67.1%	72.3%	69.6%
Losers	74.3%	69.3%	71.7%

Table 10_WEKA Naïve Baye's Results

Balanced Data Set

```
Correctly Classified Instances         434               70.684  %
Incorrectly Classified Instances       180               29.316  %
Kappa statistic                          0.4137
Mean absolute error                      0.3194
Root mean squared error                  0.4852
Relative absolute error                 63.8776 %
Root relative squared error             97.0354 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.671    0.257    0.723      0.671    0.696      0.415    0.764     0.734     Winner
                 0.743    0.329    0.693      0.743    0.717      0.415    0.764     0.770     Loser
Weighted Avg.    0.707    0.293    0.708      0.707    0.706      0.415    0.764     0.752

=== Confusion Matrix ===

   a   b   <-- classified as
 206 101 |   a = Winner
  79 228 |   b = Loser
```

Imbalanced Data Set

```
Correctly Classified Instances         468               76.2215 %
Incorrectly Classified Instances       146               23.7785 %
Kappa statistic                          0.431
Mean absolute error                      0.2734
Root mean squared error                  0.4522
Relative absolute error                 68.2041 %
Root relative squared error            101.0656 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.641    0.191    0.562      0.641    0.599      0.433    0.761     0.567     Winner
                 0.809    0.359    0.855      0.809    0.831      0.433    0.761     0.883     Loser
Weighted Avg.    0.762    0.312    0.774      0.762    0.767      0.433    0.761     0.795

=== Confusion Matrix ===

   a   b   <-- classified as
 109  61 |   a = Winner
  85 359 |   b = Loser
```

Support Vector Machine

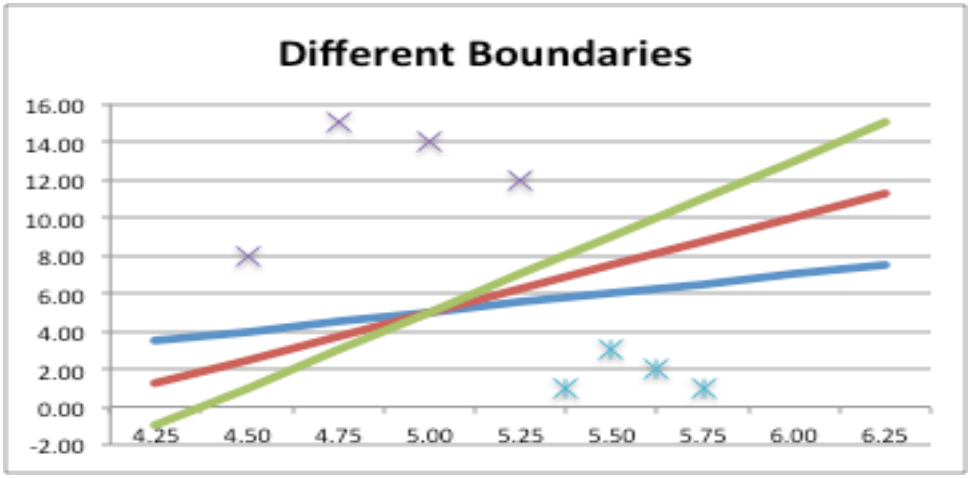
We learned earlier that the Perceptron Model finds a linear boundary that divides our trades into Winners and Losers. Support Vector Machines (SVMs) work similarly although this model can be used to find a non-linear boundary. When determining the optimal feature weights, the model focuses on those samples that are the most difficult to separate; these samples are called support points and the model maximizes the margin between these points.

Vapnik and Cortes developed the SVM model in 1992 using Vapnik and Chervonenkis' statistical learning theories from the 1960s. The model has been successfully applied to handwriting and speech recognition as well as protein sequencing and breast cancer diagnosis.

Mathematical Description

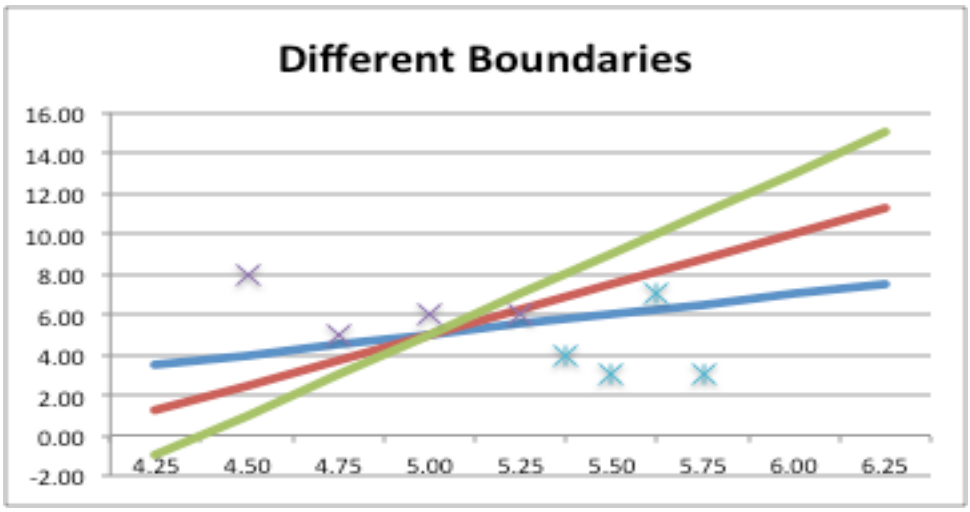
In the case of linearly separable data, many lines can accurately separate the data. In the chart below, three lines divide the data perfectly into their respective classes.

Figure 9_SVM Different Boundaries



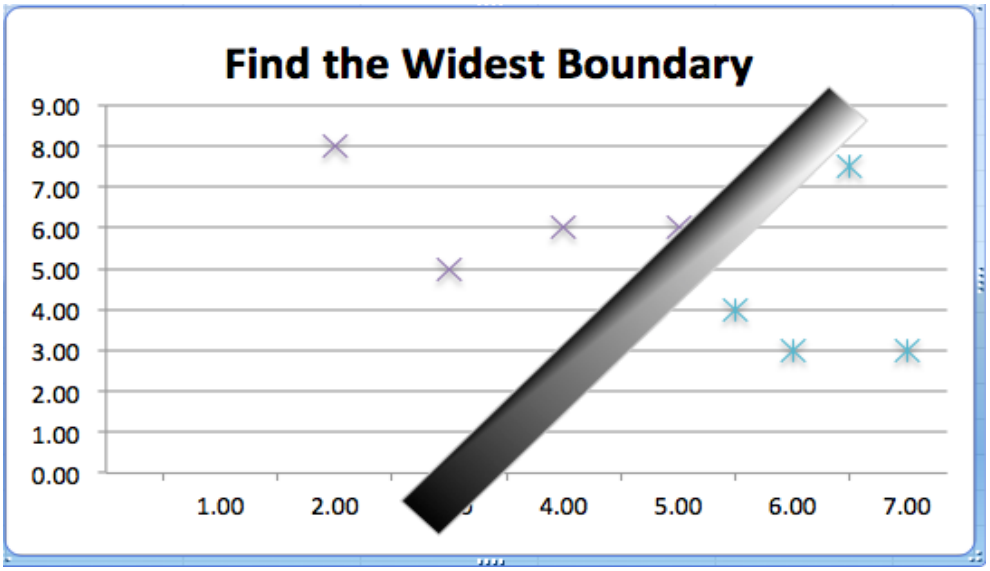
The data in the graph below, however, is not obviously separable.

Figure 10_SVM Non Conclusive Boundaries



The SVM model does two things differently than the Perceptron model to better accommodate data sets that are not obviously separable. Instead of finding a line, this model finds a rectangle that separates the classes without touching any of the data points. The points that touch the rectangle are called the support vectors and the distance between these points and the middle of the rectangle is called the margin. The SVM model maximizes this margin.

Figure 11_SVM Finds the Boundary with the widest margin



Just like the perceptron model, we will let x be a feature vector and w a weight vector, both of length d, and s their sum product.

$$x^i = \begin{bmatrix} x_1^i & x_2^i & \dots x_d^i \end{bmatrix}$$

$$w^i = \begin{bmatrix} w_1^i & w_2^i & \dots w_d^i \end{bmatrix}$$

$$s = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_d w_d = w^T x$$

$$by\ setting\ \ x_0 = 1, w_0\ becomes\ our\ bias$$

Like the Perceptron model we define our target class t_i as being 1 for a winner and -1 for a loser. Our boundary must satisfy the following rules[8]:

Equation 23_SVM Boundary Rules

$$w^T x \geq 0\ for\ t_i = +1$$

$$w^T x < 0\ for\ t_i = -1$$

To simplify,

$$t_i \times (w^T x) \geq 0$$

To make the boundary rules unique we will add the following constraint:

Equation 24 _SVM Constraint

$$|w^T x| = 1$$

$$t_i \times (w^T x) \geq 1\ which\ will\ be\ our\ constraint$$

Remember, our goal is to separate the data using a rectangular shaped boundary. The mid line of this boundary is where

$$w^T x = 0$$

The upper margin of this boundary is where

$$w^T x = 1$$

The lower margin of the boundary is where

$$w^T x = -1$$

The distance between the lower margin and the upper margin is :

$$\frac{2}{\|w\|}$$

Since our goal is to find the boundary with the widest margin, we will try to maximize this width or minimize its reciprocal

$$\frac{w^T w}{2}$$

To find an optimal boundary we will find the minimum of the objective function while keeping the constraint function equal to 1 or more

$$min \frac{1}{2} w^t w$$

$$t_i(w^T x_i + b) \geq 1$$

In cases where the data is almost linearly separable except for a few points that fall on either side of the boundary, we introduce slack variables. For other cases where the data is clearly non-separable we will use a kernel transformation to transform the nonlinear decision boundary into a linear decision boundary. We will use regularization to make the margin as large as possible and the slack variable as small as possible.

SVM Learning Process[9]

1. *Set the initial values of the weights.*
2. *Loop thru the entire data set for a set number of epochs*
3. *For each epoch, loop thru each sample*
4. *Set the value of the objective function or minimum threshold.*
    a. $min\frac{1}{2}w^t w$
5. *Compute the constraint function for each sample.*
    a. $t_i(w^T x_i + b) \geq 1$
6. *Use an optimization method to minimize the objective function while maintaining the constraint.*
7. *Adjust the weights*
    a. *If the sample is outside the boundary margin then make a small adjustment to the weight.*
        i. $w_{t+1} = \left(1 - \frac{1}{t}\right) \times w_t$
    b. *If the sample is on or inside the boundary then make a large adjustment to the weight. Lambda is the learning rate.*
        i. $w_{t+1} = \left(1 - \frac{1}{t}\right) \times w_t + \frac{1}{\lambda \times t} \times (w_t x)$
8. *Keep a running total of the squared error.*
9. *Iterate thru the data until the error is below a minimum threshold.*

Results

Table 11_SVM Results

Class	Accuracy	Reliability	F1 Stat
Winners	73.3%	76.3%	74.8%
Losers	77.2%	74.3%	75.7%

Table 12_WEKA SVM Results Output

Balanced Data

```
Correctly Classified Instances         462               75.2443 %
Incorrectly Classified Instances       152               24.7557 %
Kappa statistic                          0.5049
Mean absolute error                      0.2476
Root mean squared error                  0.4976
Relative absolute error                 49.5105 %
Root relative squared error             99.5084 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.733    0.228    0.763      0.733    0.748      0.505    0.752     0.693     Winner
                 0.772    0.267    0.743      0.772    0.757      0.505    0.752     0.688     Loser
Weighted Avg.    0.752    0.248    0.753      0.752    0.752      0.505    0.752     0.690

=== Confusion Matrix ===

   a   b   <-- classified as
 225  82 |   a = Winner
  70 237 |   b = Loser
```

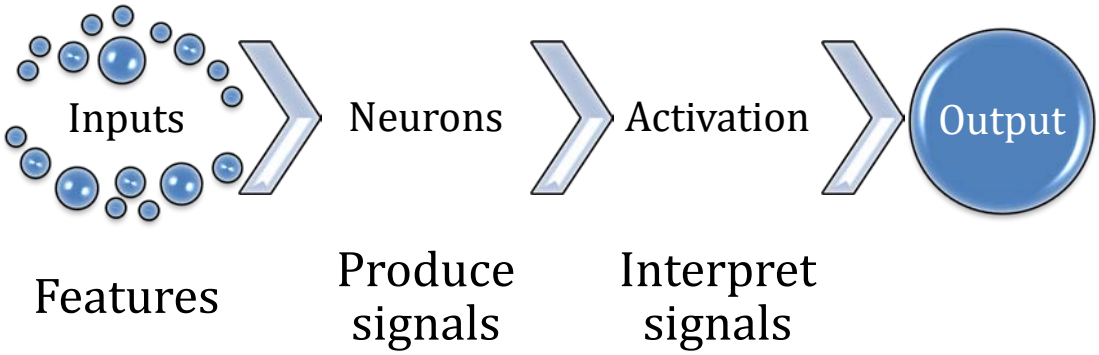
Imbalanced Data

```
Correctly Classified Instances         495               80.6189 %
Incorrectly Classified Instances       119               19.3811 %
Kappa statistic                          0.4871
Mean absolute error                      0.1938
Root mean squared error                  0.4402
Relative absolute error                 48.3573 %
Root relative squared error             98.3875 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.559    0.099    0.683      0.559    0.615      0.492    0.730     0.504     Winner
                 0.901    0.441    0.842      0.901    0.871      0.492    0.730     0.830     Loser
Weighted Avg.    0.806    0.346    0.798      0.806    0.800      0.492    0.730     0.740

=== Confusion Matrix ===

   a   b   <-- classified as
  95  75 |   a = Winner
  44 400 |   b = Loser
```

Neural Networks

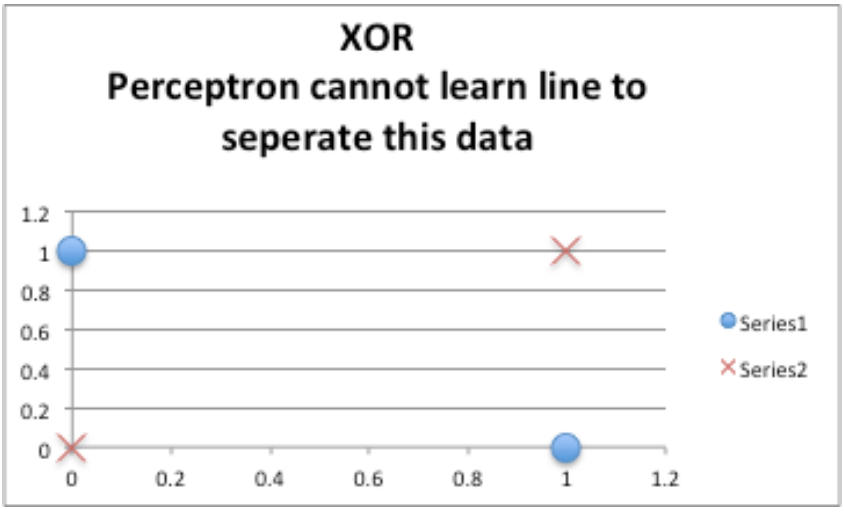
Artificial neural networks mimic the human brain function. Inputs are fed to multiple layers of neurons that produce signals that are fed to an activation cell that produces an output. Unlike the human brain, a neural network can really only accomplish one small task and it is best at pattern recognition.

Equation 25_Neural Network Diagram



After Rosenblatt introduced the concept of the perceptron in the early 1960s, Marvin Minsky published a seminal book in 1969 highlighting the limitations of the Perceptron. Specifically, its inability to learn a function that separates the data below

Equation 26_XOR Diagram



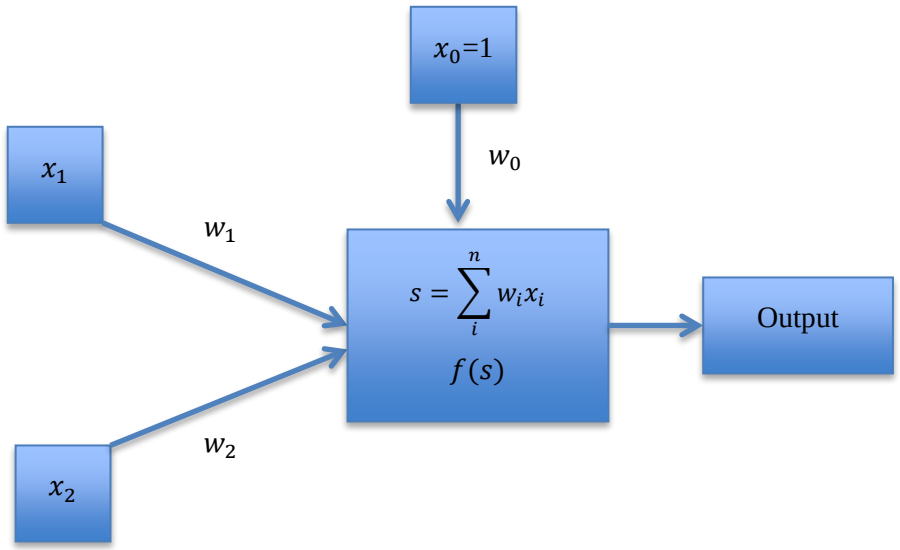
We cannot draw a straight line to separate this data. Minsky suggested that two layers of Perceptron models could successfully solve the XOR problem[10].

In 1974, Paul Werbos proposed that back propagation could be used in multilayered neural networks. In 1986, David Rumelhart, Geoffrey Hinton, and Ronald Williams published “Learning Representation by Back-Propagating Errors”.

Neural Network Mathematical Description

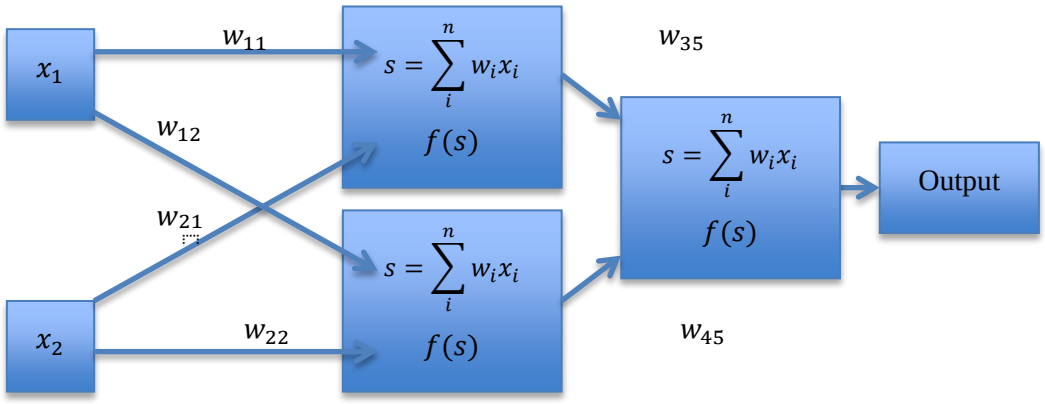
The Perceptron model is a one layer Neural Network and looks like this:

Figure 12_One Layer Neural Network Diagram



Neural Networks add another layer, which enables the algorithm to find non-linear solutions

Figure 13_Two Layer Neural Network Diagram



For feed forward networks the activation function is the same sigmoid function we used in logistic regression. In this case, a parameter β is used to adjust the output s.

$$p = \frac{1}{1 + e^{-\beta s}}$$

Neural Network Learning Process[11]

1. *Start with randomized weights.*
2. *For a predetermined number of epochs*
3. *For each epoch*
    a. *Feed forward each vector of training data and calculate the predicted target value* y_k
    b. *Compute the error of each predicted target.*
        i. $error \rightarrow e_k = t_k - y_k$
    c. *Compute the error contribution from each neuron layer.*
        i. $h_j = \sum_{k=1}^{m} g_k w_{jk}$
        ii. $g_j = h_j \times f(s_j)\left(1 - f(s_j)\right)$
    d. *Update the weights* $w_{ik} = w_{ik} + \alpha g_k x_i$ *and* $w_{0k} = w_{ok} + \alpha g_k$
4. *Do until error is less than threshold.*

Results

Table 13_Neural Network Results

Class	Accuracy	Reliability	F1 Stat
Winners	83.7%	80.8%	82.2%
Losers	80.1%	83.1%	81.6%

Table 14_WEKA Neural Network Results

Balanced Data Set

```
Correctly Classified Instances         503               81.9218 %
Incorrectly Classified Instances       111               18.0782 %
Kappa statistic                          0.6384
Mean absolute error                      0.2003
Root mean squared error                  0.3498
Relative absolute error                 40.0604 %
Root relative squared error             69.967  %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.837    0.199    0.808      0.837    0.822      0.639    0.911     0.904     Winner
                 0.801    0.163    0.831      0.801    0.816      0.639    0.911     0.906     Loser
Weighted Avg.    0.819    0.181    0.820      0.819    0.819      0.639    0.911     0.905

=== Confusion Matrix ===

   a   b   <-- classified as
 257  50 |   a = Winner
  61 246 |   b = Loser
```

Imbalanced Data Set

```
Correctly Classified Instances         507               82.5733 %
Incorrectly Classified Instances       107               17.4267 %
Kappa statistic                          0.5559
Mean absolute error                      0.2054
Root mean squared error                  0.366
Relative absolute error                 51.2376 %
Root relative squared error             81.7995 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.653    0.108    0.698      0.653    0.675      0.557    0.837     0.742     Winner
                 0.892    0.347    0.870      0.892    0.881      0.557    0.837     0.899     Loser
Weighted Avg.    0.826    0.281    0.823      0.826    0.824      0.557    0.837     0.855

=== Confusion Matrix ===

   a   b   <-- classified as
 111  59 |   a = Winner
  48 396 |   b = Loser
```

Decision Trees

The advent of computers and computer scientists in the eighties fostered a new type of machine learning. J. Ross Quinlan introduced a rule-based model in the early 70's called the Decision Tree.

He and other computer scientists continued to improve on these classification and regression tree models (CART) in the eighties. Unlike the linear models we have examined, these models revolve around "decisions" and not "coefficient weights". Both of these models are quite intuitive and perform well with non-linearly separable data.[12]

Mathematical Description

The decision tree model, also known as the classification and regression tree (cart) model, is a binary classification model. The objective is to find rules that best split target data into their correct classes.

The family game called "20 Questions" offers some insight into how these models work. In the game, one player (the leader) imagines a person, place, or thing. The remaining players try to guess what the leader is imagining and they each have 20 yes/no questions they can ask to gain clues as to what it is. The first person to guess the correct answer wins.

Similarly, the computer is given a set of sample trades that include both winners and losers. The data set includes features about each trade that offer clues about the class of each trade. When the computer asks a question about the sample data, the "yes" samples are partitioned to a left grouping and the "no" samples to a right grouping. The computer continues to ask question and partition the data until all of the samples are accurately classified as winners or losers. The questions are the rules used to classify future data. In essence, the algorithm teaches the machine how to be a professional "20-Question" player.

There are an infinite number of rules that the machine can use to split the data; the trick is figuring out which questions split the data best.

To do this we will use Information Theory and particularly the Gini Impurity index to measure the purity of each node. If all samples in a node are winners then it is a pure split and the Gini index will be zero.

The Gini index is not hard to calculate. At the node level, the Gini is equal to the weighted average of the branch level Gini indices.[13]

Equation 27_Gini Calculations

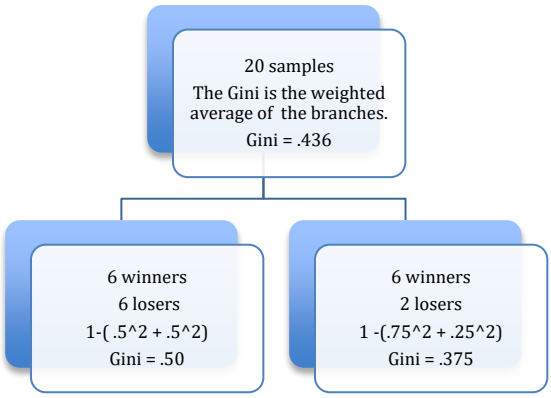
$$Gini = 1 - \sum_{k=1}^{n} p_k^2$$

$$where\ p\ is\ the\ probability\ of\ the\ target\ k$$

$$p_{winner} = \frac{count\ of\ winners\ in\ node}{count\ of\ samples\ in\ node}$$

$$p_{loser} = \frac{count\ of\ losers\ in\ node}{count\ of\ samples\ in\ node}$$

The example below describes how the Gini index is calculated at a node containing 20 samples.



To refer back to our game of "20 questions" it is always best to start off with a general question like, "are you thinking about a place?" as opposed to "are you thinking about your house at 12 Easy Street?" A more general question gives us a better idea of where to focus attention.

The “best” questions are those that result in the largest amount of information gain. Information gain is the difference between the node level Gini indices. At each branch or node, the algorithm will iterate thru all the possible features to split the data and determine the one that results in the largest amount of information gain.

Decision Tree Learning Process

1. *Calculate the Gini for every feature in the training set and select the feature with the smallest Gini as the initial Gini.*
2. *At each branch, calculate the Gini using the remaining features and identify the feature that results in the smallest Gini.*
3. *Split the data on that feature.*
4. *Repeat this process until all samples are properly classified or until a user constraint is met.*

Some of the more common user constraints include;

1. *Limit the number of splits.*
2. *Limit splits to only branches with a minimum number of samples*

The computer is a machine and without constraints it will tirelessly split the data until every training sample is classified correctly. Remember, every split adds a new “question” to the list and this list of questions is used to categorize new trade samples. If the list of questions gets too specific, it will produce great results on the training data but not so well on new data. In our case, we limited model to 20 splits and minimum leaf size to 3 samples. In the next section, we will discuss a method that will help alleviate the problem of over fitting with Decision Trees.

Decision Tree Results

Table 15_Decision Tree Results

Class	Accuracy	Reliability	F1 Stat
Winners	89.3%	84.6%	86.8%
Losers	83.7%	88.6%	86.1%

We used the balanced data set to test this model.

Table 16_WEKA Output (max 20 depth and min 3 leaf)

```
Correctly Classified Instances         531               86.4821 %
Incorrectly Classified Instances        83               13.5179 %
Kappa statistic                          0.7296
Mean absolute error                      0.1883
Root mean squared error                  0.3353
Relative absolute error                 37.6613 %
Root relative squared error             67.0648 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.893    0.163    0.846      0.893    0.868      0.731    0.912     0.887     Winner
                 0.837    0.107    0.886      0.837    0.861      0.731    0.912     0.915     Loser
Weighted Avg.    0.865    0.135    0.866      0.865    0.865      0.731    0.912     0.901

=== Confusion Matrix ===

   a   b   <-- classified as
 274  33 |   a = Winner
  50 257 |   b = Loser
```

The diagram below is a representation of the tree used to classify our balanced data set.

Tree Diagram

Figure 14_WEKA Decision Tree Output

```
REPTree
============

BSpd < 0.06 : Loser (41/1) [15/0]
BSpd >= 0.06
|   Issue_Size < 0.52
|   |   BSpd < 0.23
|   |   |   Rating < 0.84
|   |   |   |   Amount < 0.03
|   |   |   |   |   BSpd < 0.15
|   |   |   |   |   |   BSpd < 0.13
|   |   |   |   |   |   |   Issue_Size < 0.2
|   |   |   |   |   |   |   |   Issue_Size < 0.03 : Winner (9/2) [6/2]
|   |   |   |   |   |   |   |   Issue_Size >= 0.03 : Loser (37/6) [17/0]
|   |   |   |   |   |   |   Issue_Size >= 0.2 : Winner (16/1) [5/0]
|   |   |   |   |   |   BSpd >= 0.13 : Winner (39/2) [16/4]
|   |   |   |   |   BSpd >= 0.15
|   |   |   |   |   |   Rating < 0.78
|   |   |   |   |   |   |   State=MN < 0.5 : Loser (3/0) [4/1]
|   |   |   |   |   |   |   State=MN >= 0.5 : Winner (3/0) [3/0]
|   |   |   |   |   |   Rating >= 0.78 : Loser (14/0) [4/0]
|   |   |   |   Amount >= 0.03
|   |   |   |   |   Amount < 0.77 : Loser (60/6) [30/5]
|   |   |   |   |   Amount >= 0.77 : Winner (5/0) [4/0]
|   |   |   Rating >= 0.84
|   |   |   |   State=NV < 0.5
|   |   |   |   |   State=TX < 0.5 : Winner (51/5) [26/11]
|   |   |   |   |   State=TX >= 0.5 : Loser (8/3) [2/0]
|   |   |   |   State=NV >= 0.5 : Loser (3/0) [3/0]
|   |   BSpd >= 0.23 : Winner (77/7) [50/5]
|   Issue_Size >= 0.52 : Loser (43/6) [20/8]
```

Random Forests

Decision Trees make classifications based on thresholds or splits. In two-dimensional space these thresholds can be visualized as vertical and horizontal lines. Decision Trees can handle non-linearly separable data but usually at the price of over fitting. Leo Beirman designed Random Forests in 2001 to help classification and regression tree models smooth their complicated boundaries. He did this by using an ensemble technique which synthesized the classifications of a collection of Decision Trees.[14]

Random Forest Mathematical Description

At the core of the Random Forest model are Decision Trees. Whereas the Decision Tree model uses one tree to classify data, the Random Forest model uses a portfolio of trees (a forest) to make classifications. Each tree in the forest makes a classification and counts for one vote towards the final classification. The classification with the most votes is the model's final classification. If there are 20 trees, for example, and 15 classify a winner and 5 a loser then the model will classify a winner.

In non-linear data there are many samples that are in a "gray area" or an area where all the samples are not clearly on either side of a boundary. The SVM model handled these points with slack variables; the random forest model addresses these points with a portfolio of trees, each with a different opinion or vote.

First, each tree is created using a unique set of data. Bootstrapping is a process that generates a random subset of the original data set for each tree. The number of samples in each subset is equal to the number of samples in the original dataset. However, the samples are chosen randomly with replacement. Since the randomly selected data is done with replacement, statistically only 63.2% of the original data set will be in each subset.

Second, at any given branch of each tree, the algorithm limits the number of features or rules the tree can use to split the data. Most random forest models limit this number to the square root of the total number of features in the data set. These features are chosen randomly at each branch level and if there is no information gain then new features are selected.

Random Forest Learning Process[15]

1. *Determine the number of trees in the forest.*
2. *Determine the number of split features to be used at each branch level. Default value is the square root of the total number of features.*
3. *Generate a random subset of data with replacement for each tree (bootstrapping)*
4. *Generate each tree in the forest using the process described in the Decision Tree section.*
5. *Vote for final classification.*

Random Forest Results

Table 17_Random Forest Results

Class	Accuracy	Reliability	F1 Stat
Winners	98.0%	97.4%	97.7%
Losers	97.4%	98.0%	97.7%

Table 18_WEKA Random Forest Output

```
Correctly Classified Instances         600               97.7199 %
Incorrectly Classified Instances        14                2.2801 %
Kappa statistic                          0.9544
Mean absolute error                      0.1024
Root mean squared error                  0.1731
Relative absolute error                 20.4865 %
Root relative squared error             34.627  %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.980    0.026    0.974      0.980    0.977      0.954    0.995     0.996     Winner
                 0.974    0.020    0.980      0.974    0.977      0.954    0.995     0.994     Loser
Weighted Avg.    0.977    0.023    0.977      0.977    0.977      0.954    0.995     0.995

=== Confusion Matrix ===

   a   b   <-- classified as
 301   6 |   a = Winner
   8 299 |   b = Loser
```

Chapter 8 Model Choice and Optimization

Using the F1 statistic as the yardstick, the Random Forest model is our preference.

Table 19_Summary of F1 Stats for all Models

Class	Perceptron	Logistic	Naïve Bayes	Support Vector	Neural Network	Decision Tree	Random Forest
Winners	79.1%	75.9%	69.6%	74.8%	82.2%	86.8%	97.7%
Losers	79.5%	76.9%	71.7%	75.7%	81.6%	86.1%	97.7%

As we learned from the previous sections, each algorithm we evaluated had adjustable variables. To name just a few, Perceptrons and Neural Networks have a "learning rate", the Support Vector Machines have "slack variables", and Decision Trees have "leaf size".

The Random Forest has variables as well and small changes in them can significantly change the model's output. In the first experiment, we will vary the depth of each tree in the forest. The output below shows the accuracy of the model using various depths: 1) no restrictions 2) twenty levels 3) ten levels, 4) five levels and 5) two levels. The output below suggests that setting the depth at 20 is optimal.

Table 20_WEKA Random Forest Experiment – Depth

```
Dataset                   (1) trees.Ra | (2) trees (3) trees (4) trees (5) trees
----------------------------------------------------------------------------------
'pre_processed_trading_da(100)   97.20 |   97.22     94.85 *   81.66 *   73.83 *
----------------------------------------------------------------------------------
                              (v/ /*) |   (0/1/0)   (0/0/1)   (0/0/1)   (0/0/1)


Key:
(1) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
(2) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1 -depth 20' 1116839470751428698
(3) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1 -depth 10' 1116839470751428698
(4) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1 -depth 5' 1116839470751428698
(5) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1 -depth 2' 1116839470751428698
```

In the second experiment, we will vary the number of features or rules used to split the data at each node point. The output below shows the accuracy using the following rule restrictions 1) forty rules 2) twenty rules 3) ten rules 4) five rules and 5) two rules.

Table 21_WEKA Random Forest Experiment - Rule

Dataset	(1) trees.Ra	(2) trees	(3) trees	(4) trees	(5) trees
'pre_processed_trading_da(100)	97.54	97.51	97.28	96.99	96.91
	(v/ /*)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)

Key:
(1) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 40 -M 1.0 -V 0.001 -S 1 -depth 20' 1116839470751428698
(2) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 20 -M 1.0 -V 0.001 -S 1 -depth 20' 1116839470751428698
(3) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 10 -M 1.0 -V 0.001 -S 1 -depth 20' 1116839470751428698
(4) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 5 -M 1.0 -V 0.001 -S 1 -depth 20' 1116839470751428698
(5) trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 2 -M 1.0 -V 0.001 -S 1 -depth 20' 1116839470751428698

In the final experiment, we will set the depth size and feature restrictions at the optimal levels and run the model on a variety of data sets to determine which data set produces the bet results. We will test model on three training sets; 1) Imbalanced nine-feature data set 2) base case balanced nine-feature data set and 3) a balanced five-feature data set. Interestingly the balanced data set with the five most relevant features produced the best results.

Table 22_WEKA Experiment – Training Sets

Dataset	(1) trees.Rand
'pre_processed_trading_da(100)	94.34
'pre_processed_trading_da(100)	97.51
'pre_processed_trading_da(100)	97.66
(v/ /*)	

Conclusion

This paper has attempted to discuss and apply different types of machine learning models in a supervised environment. The data sets we examined consisted of trade samples and included a variety of features. We determined that a non-linear ensemble model, called Random Forest, produced far superior results. The model produced the best results when trained on a balanced data set of only the most relevant features.

Bibliography

Bishop, C. M. (2003). *Neural Networks for Pattern Recognition.* New York, NY, USA: Oxford University Press.

Brownlee, J. (2015, August 19). *8 tactics to Combat Imbalanced Classes in Your Machine Learning Dataset.* Retrieved January 5, 2017, from http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

Brownlee, J. (2013, Novemer 25). *A Tour of Machine Learning Algorithms.* Retrieved November 25, 2016, from http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Brownlee, J. (2016). *Machine Learning Algorithms from Scratch.* Jason Brownlee.

Cramer, J. (2003). *The Origins and Development of the Logit.* Abstract.

Grus, J. (2015). *Data Science From Scratch.* (M. Beaugureau, Ed.) Sebastopol, CA, USA: Oreilly Media Inc.

Hackeling, G. (2014). *Mastering Machine Learning with SciKit-Learn.* (M. Rajani, Ed.) Birmingham, UK: Packt Publishing.

Hartshorn, S. (2016). *Machine Learning With Random Forests and Decision Trees.* (F. Nerdy, Ed.) ebook.

Heaton, J. (2012). *Introduction to the Math of Neural Networks.* (W. RU.com, Ed.) Chesterfield, MO, USA: Heaton Research Inc.

Heaton, J. (n.d.). *Non-Mathematical Introduction to Neural networks.* (J. Heaton, Ed.) Retrieved from Heaton Research: http://www.heatonresearch.com/content/non-mathematical-introduction-using-neural-networks

Kumar, A. *Learning Predictive Analytics with Python.* Packt Publishing.

Morin, D. (2016). *Probability For the Enthusiastic Beginner.* Self Published throughCreate Space.

Morris, D. (2016). *Bayes' Theorem: A Visual Introduction for Beginners.* Canada: Blue Windmill Media.

Pang-Ning Tan, M. S. (2006). *Introduction to Data Mining.* (K. Hurutunian, Ed.) Boston, MA, USA: Pearson.

Raschka, S. (2015). *Python Machine Learning.* (A. Hussian, Ed.) Birmingham, UK: Packt Publishing.

Richard O. Duda, P. E. (2001). *Pattern Classification* (2nd ed.). New York, NY, USA: Wiley.

Teknomo, K. (2015). *NN Tutorial - Feedforward Network.* Revoledu.com.

Teknomo, K. (2012). *Support Vector Machines Tutorial.* Revoledu.com.

Trevor Hastie, R. T. (2001). *The Elements of Statistical Learning.* New York, NY, USA: Springer.

Webb, A. (2007). *Statistical Pattern Recognition* (2nd ed.). West Sussex, UK: Wiley.

Wikepedia. (n.d.). *Perceptron*. Retrieved January 7, 2017, from https://en.wikipedia.org/wiki/Perceptron

Citations

1. [1] (Brownlee, 8 tactics to Combat Imbalanced Classes in Your Machine Learning Dataset, 2015)
2. [2] (Brownlee, A Tour of Machine Learning Algorithms, 2013)
3. [3] (Wikepedia)
4. [4] (Cramer, 2003)
5. [5] (Brownlee, A Tour of Machine Learning Algorithms, 2013)
6. [6] (Morris, 2016)
7. [7] (Brownlee, Machine Learning Algorithms from Scratch, 2016)
8. [8] (Teknomo, Support Vector Machines Tutorial, 2012)
9. [9] (Teknomo, Support Vector Machines Tutorial, 2012)
10. [10] (Heaton, Non-Mathematical Introduction to Neural networks)
11. [11] (Teknomo, NN Tutorial - Feedforward Network, 2015)
12. [12] (Hartshorn, 2016)
13. [13] (Hartshorn, 2016)
14. [14] (Hartshorn, 2016)
15. [15] (Hartshorn, 2016)